Analisi quicksort

Nel mio progetto di laboratorio ho provato ad utilizzare 4 differenti procedure di partition:

- **`last_partition`**: la procedura di partition classica, utilizzando l'ultimo elemento come pivot.
- **`randomized_partition`**: una procedura di partition in cui si usa come pivot un elemento scelto a caso.
- **`hoare_partition`**: l'algoritmo di partition originario ideato da C. A. R. Hoare.
- **`first_partition`**: un algoritmo analogo a `last_partition`, che però utilizza il primo elemento come pivot.

Come prima cosa presenterò le prestazioni di questi algoritmi e successivamente farò alcune osservazioni.

Prestazioni algoritmi

Misurazioni su field 2 (int) in secondi:

Numero record	`last_partition`	`randomized_partition`	`hoare_partition`
10 000	0.0156	0.0000	0.0000
100 000	0.0469	0.0469	0.0313
1 000 000	0.6094	0.6718	0.4375
10 000 000	7.5937	7.3281	5.0625
20 000 000	16.8125	17.1718	10.8585

Le misurazioni sul field 3 (double) sono approssimabili a quelle sul field 2.

Invece, le misurazioni sul field 1 (string) sono decisamente peggiori, in seguito proverò a dare una mia spiegazione del motivo per cui l'ordinamento su questo campo è meno performante.

Analisi algoritmi

Se l'array da ordinare è già ordinato (o quasi) scegliere il primo (o l'ultimo) elemento come pivot può essere una cattiva idea. Quindi `last_partition` e `first_partition` possono avere delle prestazioni pessime quando gli passiamo come input un'array già ordinato.

Quando invece il pivot viene scelto casualmente (`randomized_partition`) si ha il vantaggio di ridurre le possibilità di ricadere nel caso precedente.

Paragonando `last_partition` e `randomized_partition` mi sono accorta che hanno tempi di esecuzione pressochè paragonabili, sebbene ci si aspetterebbe che `randomized_partition` sia più performante. Questo può essere causato dalla riga di codice:

```
int pivotIndex = rand() % (last - first + 1) + first;
```

Per i dati casuali infatti, la scelta del pivot non fa molta differenza e il sovraccarico della scelta di un pivot casuale è probabilmente parte del motivo per cui `randomized_partition` è leggermente più lento di quanto mi aspettassi.

La partition che si aggiudica il primo posto è sicuramente `hoare_partition`. Sebbene non sia molto intuitiva, ha il vantaggio di avere un numero minore di swap. La partition di Hoare non scambia nullla se non è necessario.

La differenza è significativa in alcuni scenari. Per esempio nel caso in cui si utilizza un ambiente in cui lo scambio o l'assegnazione di variabili/oggetti è un'operazione costosa. Ad esempio, se si usa C/C++ con matrici di oggetti. Un altro esempio tipico in cui l'implementazione della partizione di Hoare funziona meglio è quando molti degli elementi nell'array hanno lo stesso valore o quando l'array è quasi ordinato e deve solo scambiare alcuni elementi. In quei casi la versione di Hoare non esegue quasi nessuno scambio.

È questo il caso dell'ordinamento sul campo string (field 1). Inizialmente non riuscivo a capire perchè `last_partition` fosse così tanto peggiore di `hoare_partition`, però guardando al dataset utilizzato ho notato che sono presenti molte parole ripetute nel field 1.

2	12764836	'Andate	341503	24004051610
3	17178953	'Andate	2262494	88292947220
4	19822218	'Andate	1643617	65693720271
5	15467287	'Andate	1783482	5315909097
6	9904525	'Andate	2601499	92421832805
7	19074813	'Andate	2054992	46362919387
8	19074809	'Andate	1227040	34748966621
9	5077348	'Andate	1778561	18377722278
10	6829627	'Andate	1043439	58992996061
11	19074172	'Andate	362277	71171167551
12	14620447	'Andate	3211371	50118212340
13	17179546	'Andate	3417156	99407798986
14	17179769	'Andate	1503494	98873605970
15	6826418	'Andate	4783429	93989729359
16	17176056	'Andate	152236	87332350958

733161	13267153	ambedue	2098950	81403583605
733162	8363277	ambedue	987785	37492137925
733163	2057561	ambedue	987978	7829714793
733164	19314545	ambedue	821097	64901858075
733165	474602	ambedue	2425996	65770270723
733166	5702776	ambedue	1292140	33613133836
733167	19647549	ambedue	3162978	13537264929
733168	14553279	ambedue	4402902	33982003142
733169	15769386	ambedue	3331985	42348768533
733170	14462888	ambedue	2559706	92622967304
733171	5140820	ambedue	2436230	53925971069
733172	13809386	ambedue	4222540	68969859010
733173	87398	ambedue	1952404	24858013912
733174	19161024	ambedue	3203699	82043453928
733175	17814540	ambedue	2815658	18935050327

Numero record	`last_partition`	`randomized_partition`	`hoare_partition`
100 000	0.3438	0.3438	0.0781
1 000 000	35.0600	35.0156	0.7188
10 000 000	---	---	10.1257
20 000 000	---	---	24.6406

Eseguendo l'ordinamento sul campo stringa prima con `last_partition` e poi con `hoare_partition` si nota subito di come hoare sia più performante: con un campione di 1000000 record `last_partition` ci impiega circa 35 secondi, mentre `hoare_partition` ci impiega approssimativamente 0,72 secondi. Questa differenza così grande è dovuta probabilmente al fatto che il partizionamento di Hoare effettua pochi scambi quando molti degli elementi nell'array hanno lo stesso valore.